University of Southern California

Unsupervised Learning of Relational Patterns

Wei-Min Shen and Bing Leng

USC/Information Sciences Institute

January 1996

ISI/RR-96-434

INFORMATION SCIENCES INSTITUTE

310/822-1511

4676 Admiralty Way/Marina del Rey/California 90292-6695

Unsupervised Learning of Relational Patterns

Wei-Min Shen and Bing Leng

USC/Information Sciences Institute

January 1996

ISI/RR-96-434

REPORT DOCUMENTATION PAGE

FORM APPROVED
OMB NO. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE January 1996	3. REPORT TYPE AND DATES COVERED Research Report

4. TITLE AND SUBTITLE

Unsupervised Learning of Relational Patterns

5. FUNDING NUMBERS

Rome Labs/ARPA:
F30602-94-C-0210
and
MDA972-94-2-0010
CA: C94-0031
Motorola/ECC: None

6. AUTHOR(S)

Wei-Min Shen and Bing Leng

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

USC INFORMATION SCIENCES INSTITUTE
4676 ADMIRALTY WAY
MARINA DEL REY, CA 90292-6695

8. PERFORMING ORGANIZATON REPORT NUMBER

ISI/RR-96-434

9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)

ARPA	Motorola, Inc.	Rome Labs	Eastman Chemical Co.
3701 N. Fairfax Dr.	5918 W.Courtyard Dr.	Griffiss AFB	Lincoln Street
Arlington, VA 22203	Austin,TX 78730	NY 13441	Kingsport, TN 37662

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12A. DISTRIBUTION/AVAILABILITY STATEMENT

UNCLASSIFIED/UNLIMITED

12B. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Learning relational patterns without supervision is a challenging and open problem for machine learning, yet a very desirable feature for real-world applications. In this paper, we present a new algorithm for unsupervised learning of relational patterns. Given a relational database, this algorithm can find function-free Horn clauses without requiring users to label the data as positive or negative examples, and specify what target concepts to learn. The algorithm is a heuristic search through the relational pattern space. It starts with general patterns derived from a given database schema, and then iteratively generates new promising patterns using the knowledge dynamically collected in the learning process, such as previously learned patterns. To demonstrate the capability of the algorithm, we show some abstracted database examples, and we also show that some of the well-known examples in the relational concept learning literature can be learned without supervision.

14. SUBJECT TERMS

algorithms, data mining, relational patterns, unsupervised learning

15. NUMBER OF PAGES

17

16. PRICE CODE

17. SECURITY CLASSIFICTION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UNLIMITED

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reoprts. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to stay within the lines to meet optical scanning requirements.

Block 1. Agency Use Only (Leave blank).

Block 2. Report Date. Full publication date including day, month,a nd year, if available (e.g. 1 jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element numbers(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C	- Contract	PR	- Project
G	- Grant	TA	- Task
PE	- Program Element	WU	- Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the repor.

Block 9. Sponsoring/Monitoring Agency Names(s) and Address(es). Self-explanatory

Block 10. Sponsoring/Monitoring Agency Report Number. (If known)

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of ...; To be published in... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

DOD	- See DoDD 5230.24, "Distribution Statements on Technical Documents."
DOE	- See authorities.
NASA	- See Handbook NHB 2200.2.
NTIS	- Leave blank.

Block 12b. Distribution Code.

DOD	- Leave blank.
DOE	- Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
NASA	- Leave blank.
NTIS	- Leave blank.

Block 13. Abstract. Include a brief (Maximum 200 words) factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code (NTIS only).

Blocks 17.-19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contins classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.

Report Documentation Page, Form SF298, Continued

9. Sponsoring/Monitoring Agency Name(s) and Addresses, continued

Office of Strategic Technology
California Trade and Commerce Agency
200 East Del Mar Blvd, Suite 204
Pasadena, CA 91105
(818) 568-9437

Unsupervised Learning of Relational Patterns

Abstract

Learning relational patterns without supervision is a challenging and open problem for machine learning, yet a very desirable feature for real-world applications. In this paper, we present a new algorithm for unsupervised learning of relational patterns. Given a relational database, this algorithm can find function-free Horn clauses without requiring users to label the data as positive or negative examples, and specify what target concepts to learn. The algorithm is a heuristic search through the relational pattern space. It starts with general patterns derived from a given database schema, and then iteratively generates new promising patterns using the knowledge dynamically collected in the learning process, such as previously learned patterns. To demonstrate the capability of the algorithm, we show some abstracted database examples, and we also show that some of the well-known examples in the relational concept learning literature can be learned without supervision.

1 Introduction

Relational patterns are important. They are more expressive than attribute-value pair representation, more nature to read when describing complex objects and concepts, and have the power to predict unknown values or relations. For instance, the pattern

$$ancestor(X,Y), parent(Y,Z) \Rightarrow ancestor(X,Z)$$

describes naturally an interesting relationship between objects X, Y, and Z, which is hard to represent using attribute-value pairs. In fact, this recursive relation parsimoniously represents a very large (if not infinity) set of extensional data points, and can predict that X is an ancestor of Z when there is an object Y such that the objects X, Y, and Z satisfy its left-hand side.

To discover such relational patterns from real-world data sets and databases, unsupervised learning is necessary. Since the examples of relational patterns are composed of multiple tuples from different database tables and the sizes of databases are normally very large, it is impractical, in some cases even impossible, to label the examples. Moreover, sometimes we even don't know a priori what patterns to learn.

It is unfortunate that current machine learning systems have limitations for this type of learning. On the one hand, relational concept learning systems need supervision. Most existing systems such as FOIL[10], FOCL[9], CIGOL[7], Progol[8] need pre-specified concepts and pre-labeled examples. On the other hand, unsupervised learning systems mainly focus on attribute-value concepts. Only a few of them, such as CLUSTER/S[13], MOBAL[15], KLUSTER[5], LABYRINTH[14], and KBG[2,3], can learn more expressive concepts. However, they focused on different aspects than ours (a detailed analysis is given in the related work section).

The learning task we are interested in can be summarized as follows: Given a relational database along with its schema, a learning system must find relational patterns, represented as function-free Horn clauses, that satisfy some user specified significance (or interestingness) thresholds. For example, given a database with schema: parent(p1 string, p2 string), and ancestor(a1 string, a2 string), the learning system to be designed will output relational patterns like:

```
              parent(X,Y) => ancestor(X,Y) [0.2 0.8]
ancestor(X,Z), parent(Z,Y) => ancestor(X,Y) [0.3 0.9]
```

where the two numbers (their definitions will be given later) associated with each pattern is the significance measurement of that pattern. A pattern is accepted by the system only when its significance is above the pre-specified thresholds.

In order to accomplish this task, the learning system to be designed must connect to real databases, deal with noise in the databases, control the search complexity, and use available knowledge. In the following sections, we will describe such an algorithm which addresses these issues.

2 The Δ-Algorithm

The Δ-algorithm described in this section is an algorithm that meets the requirements specified above. Given a relational database, the schema of the database, three user-specified significance thresholds, and optionally some domain knowledge if available, this algorithm will output significant relational patterns in an incremental fashion.

The algorithm is a generate-and-test process that consists of three main phases. It first finds connections (to be defined below) between database tables. From these connections, it then generates all possible transitivity[1] patterns and tests them against the database to see if they have high enough significance. Those patterns that past the tests will be returned to the users, and those that partially fail the tests will be recorded as plausible patterns for further search. The reason we choose transitivity patterns as a focal point here is because these patterns are highly structured and they are general enough to derive other types of pattern, such as implication, inheritance, transfer-through, and function dependency (see [11] for details). Finally, based on the recorded plausible transitivity patterns, the algorithm

[1]A transitivity pattern is a pattern $LHS \Rightarrow RHS$ that relates a set of objects, $A_1, A_2, ..., A_n$, in such a way that its left-hand side LHS specifies the consecutive relations from A_1 to A_n, and its right-hand side RHS specifies a single relation between A_1 and A_n. The ancestor pattern, $ancestor(X,Y) \wedge parent(Y,Z) \Rightarrow ancestor(X,Z)$, is an example of transitivity pattern.

```
Inputs: A relational database and its schema, three thresholds $c$, $b$, and $s$ (between 0 and 1);
Outputs: Relational patterns that have significance no less than $b$ and $s$;
Procedure:
     /* Phase 1: Find connections between tables */
1.   Identify a set $X$ of pairs of data fields that are significantly connected with respect to $c$,
2.   Construct the significant connection table (SCT) from $X$,
3.   Convert SCT to a corresponding graph $G$;
     /* Phase 2: Generate and evaluate transitivity patterns */
4.   For $k = 2$ to $|G|$ do,
5.      Generate from $G$ a set $P$ of transitivity patterns with length $k$,
6.      For each $p$ in $P$,
7.         Computer the base value $p_b$ and the strength value $p_s$,
8.         When $p_b \geq b$,
9.            If $p_s \geq s$, then output $p$,
10.           else record $p$ in a set $Q$ of plausible patterns;
     /* Phase 3: Generate and evaluate other types of patterns */
11.  Repeat:
12.     Select $q$ from $Q$,
13.     Expand $q$ by selecting and adding a constraint $r$ to its left-hand side,
14.     Compute the base value $q_b$ and the strength value $q_s$,
15.     When $q_b \geq b$,
16.        If $q_s \geq s$, then output $q$,
17.        else insert $q$ into $Q$,
18.  Until $Q$ is empty.
```

Figure 1: The Δ-Algorithm

searches for more sophisticated patterns by adding constraints to these patterns and testing their significance against the database.

The entire Δ-algorithm is listed in Figure 1. We shall point out that the control structure between phase two and three can be flexible. We can either do them separately, as shown in Figure 1, or interleavingly, by moving Lines 11 through 18 into the loop started at Line 6. The second option is useful when the size of the graph G is very large and users are interested in seeing shorter patterns first (this bears the same philosophy as [6]). In the following sections, we shall explain each phase in detail.

2.1 Find Connections Between Database Tables

In order to generate relational patterns, we first identify sets of data fields (also known as "columns" in the database literature) that are significantly connected. Two columns, from different database tables, are significantly connected, if they have the same type and have ranges that overlap each other above the user specified threshold c, where $0 < c < 1$. The degree of overlapping is computed as follows. Let C_x and C_y be two columns, and V_x and V_y be their value sets, respectively, then the overlapping of C_x and C_y is the maximum number of the shared values relative to either V_x or V_y, as follows:

$$Overlap(C_x, C_y) = \max(\frac{|V_x \cap V_y|}{|V_x|}, \frac{|V_x \cap V_y|}{|V_y|}).$$

Here domain knowledge may be used to eliminate unnecessary connections (e.g., height vs. temperature) or suggest and establish syntactically different connections (e.g., color vs. light frequency). Each pair of columns that are connected are then given a reference name, and these connections will be represented in a significant connection table (SCT), where each row is a connection, each column is a table, and each non-empty entry is the name of a connected data field (or column).

```
T1:(C11 C12 C13)  T2:(C21 C22 C23)  T3:(C31 C32)  T4:(C41 C42 C43)
----------------  ----------------  ------------  ----------------
   (jj  5   0.5)     (14  0.5 mmm)     (14  oo)      (nnn 0.0 5)
   (nn  5   0.8)     (14  0.6 iii)     (15  kk)      (mmm 0.0 6)
   (ll  7   0.5)     (14  0.3 jjj)     (16  mm)      (rrr 0.1 4)
   (qq  5   0.5)     (12  0.7 nnn)     (15  kk)      (mmm 0.0 4)
   (kk  5   0.6)     (12  0.1 lll)     (16  ll)      (ooo 0.1 7)
   (pp  4   0.6)     (15  0.6 ppp)     (15  ll)      (jjj 0.0 4)
   (mm  2   0.5)     (15  0.4 mmm)     (15  mm)      (kkk 0.0 5)
   (nn  4   0.6)     (13  0.6 ooo)     (13  oo)      (mmm 0.0 5)
   (kk  4   0.4)     (16  0.6 ooo)     (16  oo)      (jjj 0.1 4)
   (nn  5   0.4)     (17  0.4 mmm)     (14  mm)      (mmm 0.0 5)
                     (14  0.4 lll)     (13  mm)      (jjj 0.0 5)
                     (14  0.6 kkk)     (14  mm)      (jjj 0.0 5)
                     (15  0.3 mmm)                   (lll 0.0 7)
                     (12  0.5 mmm)                   (nnn 0.1 4)
                     (15  0.4 nnn)
                     (15  0.6 ooo)
                     (16  0.5 ppp)
                     (16  0.7 ppp)

Its schema and value ranges:
------------------------------------------------------------------
T1: C11 char(2)           C12 integer[2-7]        C13 float[0.4-0.8]
T2: C21 integer[12-17]    C22 float[0.1-0.7]      C23 char(3)
T3: C31 integer[13-16]    C32 char(2)
T4: C41 char(3)           C42 float[0.0-0.1]      C43 integer[4-7]
-------------------------------------------------------------------
```

Figure 2: An abstract database

To illustrate the idea, consider for example an abstract database and its schema shown in Figure 2. In this database, there are four tables, T1 to T4, each has some columns C_{ij}.

Table 1: The Significant Connection Table for the abstracted database

	T1	T2	T3	T4
X1	C13	C22		
X2	C11		C32	
X3	C12			C43
X4		C21	C31	
X5		C23		C41

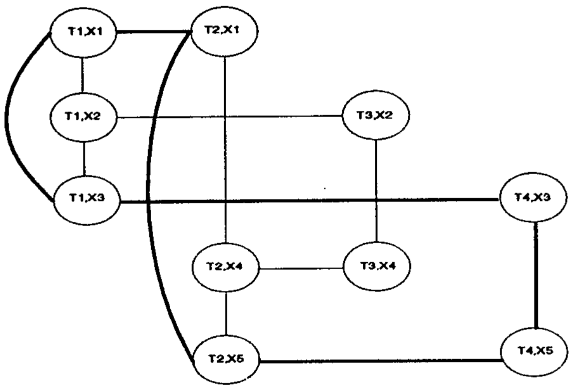

Figure 3: The corresponding graph of the abstract database's SCT

For simplicity, the value ranges of each column are listed along with the schema. (In reality, value ranges can be obtained by simple SQL queries.)

Given these information, pairs of columns that are connected can be easily determined according to our definition. For example, suppose the threshold c for overlapping is set to 0.6, then column C13 in table T1 and column C22 in table T2 are connected because they have the same data type and their overlapping is 0.9. A reference name, X1, is then created for this pair of connected columns. After considering every pair of columns, a significant connection table, shown in Table 1, is constructed. As we can see, every connected pair of columns is represetned as a row in this SCT. For instance, columns C13 and C22 are in the first row, where C13 is under T1 while C22 is under T2.

For reasons that will become clear later, we also represent the information in SCT as a graph G, where each node in G is an non-empty entry in the SCT, and each edge connects two non-empty entries that are on the same row or column in the SCT. For example, the graph built from the SCT in Table 1 is shown in Figure 3, where node (T1,X1) and node (T2,X1) represent two non-empty entries, C13 and C22, in the SCT. Since they are in the same row, there is an edge between them. Similarly, node (T1,X1) and node (T1,X2) represent two non-empty entries, C13 and C11, in the SCT, this time they are on the same column, so there is also an edge between them.

2.2 Generate Transitivity Patterns

In the second phase of Δ-algorithm, the graph G generated above provides a basis for generating all possible transitivity patterns in a given database. The idea is to find all the circles in the graph with alternated vertical and horizantal edges, and convert each of these circles into a "circle" of predicates before generating a set of transitivity patterns.

Finding circles in a graph can be accomplished by using a standard transitive closure algorithm (see for example [1]) with some simple augmentations to enforce the alternating edge constraint. A graph G cannot have more than $|G|!$ circles because a circle, without duplicated nodes, cannot be longer than the size of the graph. To convert a cirlce of graph nodes into a circle of predicates is also a straightforward task; one can simply rewrite each verticle edge in the circle by the table name. For example, the circle indicated by thick lines in Figure 3:

(T2,X1)(T2,X5)(T4,X5)(T4,X3)(T1,X3)(T1,X1)*(T2,X1)*

can be rewritten as a circle of predicates as follows:

T2(X1,X5), T4(X5,X3), T1(X3,X1),

where T2(X1,X5) is a rewrite of the verticle edge (T2,X1)(T2,X5), and T4(X5,X3) is a rewrite of (T4,X5)(T4,X3), and so on. Using this method, we can generate all circles of predicates from the graph G. In our current example, from the graph in Figure 3, all circles of predicate are found as follows[2]:

```
T2(X1 X5) T4(X3 X5) T1(X1 X3)
T2(X1 X4) T3(X2 X4) T1(X1 X2)
T2(X5 X1) T1(X1 X2) T3(X2 X4) T2(X4 X5)
T2(X4 X5) T4(X5 X3) T1(X3 X1) T2(X1 X4)
T1(X3 X1) T2(X1 X4) T3(X4 X2) T1(X2 X3)
T3(X2 X4) T2(X4 X5) T4(X5 X3) T1(X3 X2)
T1(X2 X3) T4(X3 X5) T2(X5 X1) T1(X1 X2)
T2(X1 X4) T3(X4 X2) T1(X2 X3) T4(X3 X5) T2(X5 X1)
T1(X1 X2) T3(X2 X4) T2(X4 X5) T4(X5 X3) T1(X3 X1)
```

Each circle of predicates can be used to generate a set of transitivity Horn clauses patterns by taking each predicate as the right-hand side of the pattern and the rest as the left-hand side. For instance, the first circle in the above list can produce three transitivity patterns:

```
T2(X1 X5) T4(X5 X3) => T1(X1 X3)
T1(X3 X1) T2(X1 X5) => T4(X3 X5)
T4(X5 X3) T1(X3 X1) => T2(X5 X1)
```

2.3 Evaluate Patterns

Not all patterns generated will be supported by the data in the database; their significance must be evaluated and compared to user specified thresholds. Each pattern p will be evaluated by two values that are computed against the databases: the base value p_b which reflects

[2]Note that the order of columns in a table (predicate) is not significant in a relational database.

how much the left-hand side of p is supported by the actual data in the database, and the strength value p_s which reflects how much the right-hand side is supported by the data satisfying the left-hand side.

The base value is computed as

$$p_b = \frac{|LHS|}{|DOM(LHS)|}$$

where LHS is the set of tuples in the database for which the left-hand side of p is true ($|LHS|$ is the cardinality of LHS), and $DOM(LHS)$ is the *domain* of the tables in the left-hand side. A domain of two or more tables is defined as the union of the values of the fields through which these tables are joined. For example, if two tables S and T are joined by fields $S.a$ and $T.b$, and the field $S.a$ has values $\{v_1, v_2, v_3\}$, and the field $T.b$ has values $\{v_1, v_2, v_4\}$, then the domain of S and T, $DOM(S.a, T.b)$, in this join is the set of $\{v_1, v_2, v_3, v_4\}$, and the cadinality of this domain is $|DOM(S.a, T.b)| = 4$.

The strength value is computed as

$$p_s = \frac{|RHS|}{|LHS|}$$

where LHS is defined the same as above, and RHS is the set of tuples in the LHS (not in the database) that satisfy the right-hand side of p.

A pattern's base and strength values, p_b and p_s, are compared with the input thresholds b and s. When both base and strength values are above their thresholds, the pattern is accepted. If the base value is the only one above its threshold, then the pattern is considered plausible. Such a pattern still has enough tuples (for it has a high enough base value) to be constrained to increase the strength value, and is kept in the set Q for further search. A pattern is discarded when both base and strength are below the thresholds.

In our current example, when the generated patterns are evaluated against the database in Figure 2, we get the following list (not all patterns are shown):

```
T1(X2 X1) T3(X4 X2) => T2(X4 X1) [0.17 0.7]
......
T3(X4 X2) T1(X2 X3) T4(X5 X3) => T2(X4 X5) [0.02 0.5]
......
T2(X4 X1) T1(X3 X1) T4(X5 X3) => T2(X4 X5) [0.15 0.4]
```

Suppose the thresholds are given as $b = 0.1$ and $s = 0.7$, then the first pattern will be accepted, the second discarded, and the third kept as a plausible pattern (in the Q set). Notice that right now the thresholds are set by users at the outset and the users may need several trial-and-errors to find suitable thresholds for a given database. In the future work section, we discuss some ways to remedy this problem.

2.4 Search for Patterns of Other Types

In the third phase of Δ-algorithm, the plausible patterns recorded in Q in the second phase will be used as basis for searching more patterns of other types by adding constraint to the left-hand side of a plausible pattern. A constraint can be a table connected (i.e., share a reference name in SCT) to at least one predicate in the pattern or a build-in predicate (e.g., =) with some variables that are already in the pattern. For example, we can add a gender constraint to the parent predicate and expand our ancestor pattern as follows:

$$ancestor(X,Y), parent(Y,Z), gender(Y, male) \Rightarrow ancestor(X,Z).$$

In general, adding a new constraint to the left-hand side of a pattern will reduce the size of LHS, thus the size of the Q set will decrease over time (because patterns that have too low base value will be discarded). Furthermore, we only consider to add a constraint to a pattern when it reduces the number of tuples that satisfy the left-hand side of the pattern, so the third phase of the algorithm is gauranteed to terminate.

Here, domain knowledge can be used to select constraints as well. For example, the domain knowledge may indicate that only certain type of constraints are meaningful for certain patterns, so the system can avoid considering much unnecessary search. The new pattern generated will be evaluated by the same method used in the second phase. The search stops when the set Q is empty, that is, there is no more plausible pattern left.

3 Demonstration with Examples

A prototype of the Δ-algorithm has been implemented, and in this section, we will demonstrate the capability of the algorithm by showing that some of the well-known examples in the relational concept learning literature can be learned without supervision.

3.1 A Small Network Example

The first example is a small network example used by Quinlan in his FOIL system. Using this example, Quinlan has shown that given a concept "Can-reach", and its positive and negative examples, FOIL can learn recursive definition for the concept. We, however, will use the same example, with no pre-specified concept and pre-labeled examples, to learn the same concept definition and possibly, other concept definitions as well.

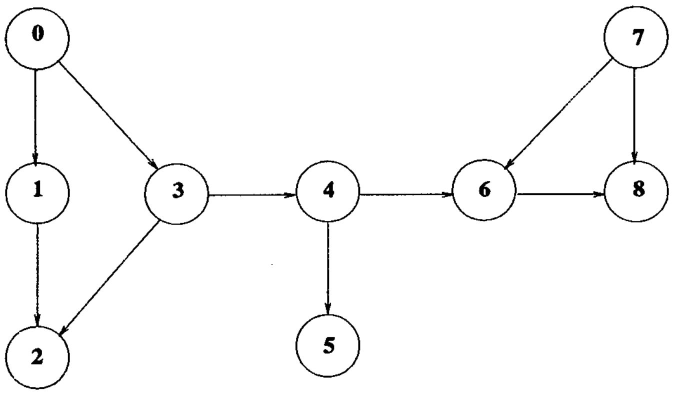

Figure 4: A small network example

The network is shown in Figure 4 and there are nine nodes, labeled from 0 to 8, and ten directed edges between nodes. If the network is represented in relational database tables, two tables are resolved, one is "Linked-to", the other is "Can-reach", both shown in Figure 5.

```
Linked-to: (B1    B2)  Can-reach: (A1    A2) (A1     A2)
           ----------             ---------- ----------
           (0     1)              (0     1)  (3      5)
           (0     3)              (0     2)  (3      6)
           (1     2)              (0     3)  (3      8)
           (3     2)              (0     4)  (4      5)
           (3     4)              (0     5)  (4      6)
           (4     5)              (0     6)  (4      8)
           (4     6)              (0     8)  (6      8)
           (6     8)              (1     2)  (7      6)
           (7     6)              (3     2)  (7      8)
           (7     8)              (3     4)
Its schema and value ranges:
---------------------------------------------
Can-reach: A1 integer[0-7]     A2 integer[1-8]
Linked-to: B1 integer[0-7]     B2 integer[1-8]
---------------------------------------------
```

Figure 5: The network database

Both tables have two columns recording the nodes in the network. An row in the table "Linked-to" records a pair of nodes connected by a single arrow in the network. For example, the row (0 1) represents the fact that node 0 is connected to node 1 by a single arrow. A row in table "Can-reach" records a pair of nodes connected by a sequence of arrow(s). For example, the row (3 8) represents the fact that node 3 is connected to node 8 through a sequence of three arrows.

Given the data tables and schema, the Δ-algorithm first checks if any pair of columns, from different tables, are significantly connected. Since all the columns are the same type, every pair is considered. Given that the significant connection threshold is set to $c = 0.6$,[3] the algorithm finds out that column A1 is connected to B1, A2 is connected to B1, and A2 to B2. These connection information is then used to construct the SCT in Table 2.

From the SCT, the Δ-algorithm builds the corresponding graph as shown in Figure 6, and generates the following set of circles of predicates:

```
Linked-to(X1 X3) Can-reach(X1 X3)
Can-reach(X1 X2) Linked-to(X2 X3) Can-reach(X1 X3)
Linked-to(X3 X1) Can-reach(X1 X2) Linked-to(X2 X3)
```

[3]The threshold 0.6 is used because it is a little above average, but not too restrict. In fact, in this domain, there is no difference for what threshold to use, except that when c is set to 0.5, one redundant pattern comes back: Linked-to(X1 X2) Can-reach(X2 X4) ⇒ Can-reach(X1 X4) [0.14 1.0].

Table 2: The Significant Connection Table for the network database

	Can-reach	Linked-to
X1	A1	B1
X2	A2	B1
X3	A2	B2

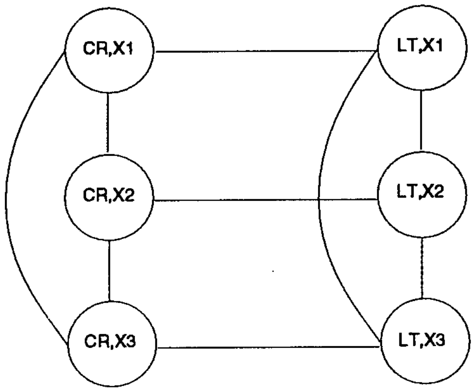

Figure 6: The corresponding graph of the network' SCT

A set of transitivity patterns are then generated and evaluated against the database. If the base value is set to $b = 0.1$ and the strength $s = 0.7$, we get the same concept as FOIL:

```
                  Linked-to(X1 X3) => Can-reach(X1 X3) [1.0  1.0]
Can-reach(X1 X2) Linked-to(X2 X3) => Can-reach(X1 X3) [0.14 1.0]
```

If we use lower threshold, e.g., $s = 0.1$, the algorithm found the following patterns as well:

```
                  Can-reach(X1 X3) => Linked-to(X1 X3) [1.0  0.5]
Can-reach(X1 X2) Can-reach(X1 X3) => Linked-to(X2 X3) [1.0  0.1]
Can-reach(X1 X3) Linked-to(X2 X3) => Can-reach(X1 X2) [0.63 0.4]
```

The pattern "Can-reach(X1 X3) $\Rightarrow$ Linked-to(X1 X3)" reflects the fact that half the entries in table "Can-reach" are the same entries in table "Linked-to". The pattern "Can-reach(X1 X2) Can-reach(X1 X3) $\Rightarrow$ Linked-to(X2 X3)" says that if two nodes can be reached from the same node (a fork shape), then they are linked by a single arrow. At this point, however, its strength, 0.1, is low. Finally, the pattern "Can-reach(X1 X3) Linked-to(X2 X3) $\Rightarrow$ Can-reach(X1 X2)" is similar to the last pattern, except focusing on two paths that have the same ending node.

3.2 A Family Tree Example

The second illustration of the Δ-algorithm is a family tree example. It was first used by Hinton in his neural network system [4], and then used by Quinlan in his FOIL system.

Again, using this example, Quinlan shown that given a concept and its positive and negative examples, FOIL can learn its relational definition. Here, we show that the concept can be learned without supervision[4].

In this example, there are two isomorphic family trees, each with twelve members, as shown in Figure 7. There are twelve relations in the trees — wife, husband, mother, father,

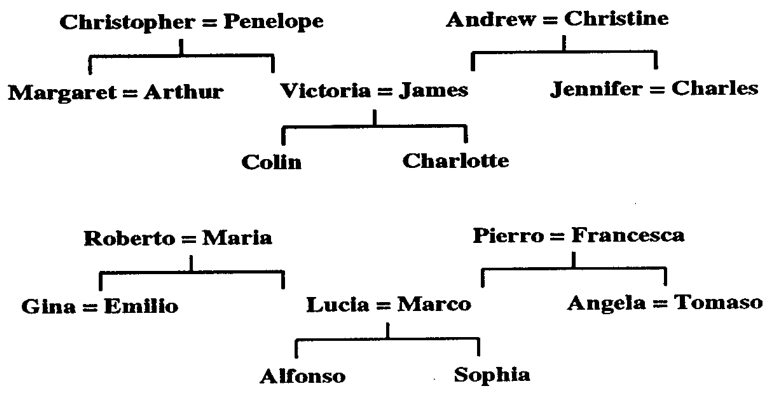

Figure 7: A family tree example, where "=" means "married to."

daughter, son, sister, brother, aunt, uncle, niece, and nephew. And two trees can be represented as twelve tables, each corresponding a relation. For example, table Wife corresponds to relation wife.

Wife:

(A1	A2)
(penelope	christopher)
(christine	andrew)
(margaret	arthur)
(victoria	james)
(jennifer	charles)
(maria	roberto)
(francesca	pierro)
(gina	emilio)
(lucia	marco)
(angela	tomaso)

The database schema and value ranges:

Wife:	A1 char(11)	A2 char(11)
Husband:	B1 char(11)	B2 char(11)
Mother:	C1 char(11)	C2 char(11)
Father:	D1 char(11)	D2 char(11)
Daughter:	E1 char(11)	E2 char(11)
Son:	F1 char(11)	F2 char(11)
Brother:	G1 char(11)	G2 char(11)
Sister:	H1 char(11)	H2 char(11)
Aunt:	I1 char(11)	I2 char(11)
Uncle:	J1 char(11)	J2 char(11)
Niece:	K1 char(11)	K2 char(11)
Nephew:	L1 char(11)	L2 char(11)

Figure 8: The family tree database

[4]Quinlan, in his paper, didn't give the actual concept definitions FOIL learned. Instead, he reported a better accuracy 78/80 (from FOIL) over 7/8 (from Hinton's network). Since our goal is to show that relational concept definitions can be learned without supervision, we will just report the concept definitions learned by the Δ-algorithm.

Once again, the algorithm goes through the same procedure to construct significant connection table, to convert the table to graph, to generate transitivity patterns from the graph, and to evaluate the patterns against the database[5]. The algorithm has found the definitions of all twelve concepts. Some, 41 patterns, are true in general, take a few for example:

```
                     Husband(X2 X1) => Wife(X1 X2)      [1.0  1.0]
   Brother(X66 X65) Niece(X65 X78) => Nephew(X66 X78) [0.22 1.0]
Daughter(X56 X57) Brother(X57 X60) => Son(X56 X60)    [0.33 1.0]
```

some, 12 patterns, are true only with respect to the given database, for instance:

```
Brother(X63 X64) => Sister(X64 X63) [1.0 1.0]
```

4 Related Work

There are some work on unsupervised learning of concepts that are beyond attribute-value pairs, such as Stepp and Michalski's CLUSTER/S, Wrobel's MOBAL, Thompson and Langley's LABYRINTH, Kietz and Morik's KLUSTER, and Bisson's KBG. Each system focus on different aspects of unsupervised relational concept learning, from the goal and context of learning, ways of learning (incremental or constructive), to the final concept form. However, no system has considered learning relational concepts (in the form of function-free Horn clauses) directly from real-world databases.

CLUSTER/S performs goal-oriented classification on structured objects by using two methods. The first is concept formation by repeated discrimination which requires both positive and negative examples. The second is concept formation by finding classifying attributes. This method attempts to find one or more classifying attributes whose value sets can be split into ranges that define individual classes. It is not clear how this method could learn the relationships among the given attributes.

MOBAL regards concept formation as a process of forming extensional aggregates of objects and then characterizing these aggregates with an intensional definition. Given a knowledge base of facts and rules that may be overly general, MOBAL uses its knowledge revision tool (KRT) to correct those overly general rules. Whenever KRT cannot correctly specialize an incorrect rule, the rule's instances and exceptions are used as positive and negative examples of a new concept. These examples are then passed to its concept learning tool (CLT) which uses a model-driven, most-general learner (RDT) to induce the concept definition. MOBAL requires some rules to start with which are not available in a database. Although RDT can learn concepts, it needs pre-labeled examples.

LABYRINTH incrementally forms concepts on composite objects, while KLUSTER constructively induces structural knowledge on term-subsumption formalisms. Both systems produce concept hierarchy which is a different representation from function-free Horn clauses.

[5]In this experiment, we iteratively generate patterns of shorter length to longer length. At time of writing this paper, we have generated patterns up to length 3. More experiments with longer length are underway, and new results will be reported.

KBG is a knowledge based generalizer in a first order logic representation. It forms generalization tree, rather than implication rules, from a set of examples, each of which is a collection of tuples, by iterative use of clustering and generalization. However, its generalization tree cannot represent recursive concepts, its examples need pre-grouped from database tables.

5 Conclusions and Future Work

We have presented an algorithm which learns relational patterns directly from databases by performing a heuristic search through the relational pattern space. It requires no pre-specified concepts and pre-labeled examples. The learned patterns have associated probabilistic significance which gives the ability to tolerant noise. The capability of the algorithm has been demonstrated by an abstracted database example and two well-known examples in the relational concept learning literature.

There are several areas need more research. The first is to show that more examples in the relational concept learning literature can be learned without supervision. Our initial analysis indicates that this is positive. The second area is to investigate more constraints on how to control the search. At the present, the only constraints we used come from the database schema, the value ranges, and the previously learned patterns. Constraints used by FOCL, such as multiple argument constraints and partial operational concept definitions, can be used as a start point. The third is to investigate both systematic and interactive ways to set the thresholds. Right now, user have to provide them at the outset, which is the weakness of the algorithm. One way to ease this problem is to start with high thresholds and gradually decrease them if too few patterns are found. The fourth is to integrate the Δ-algorithm with mining systems such as [12] to automatically generate *metapatterns* to increase the efficiency and effectiveness of today's knowlege discovery systems. Actually, this work is developed in that context. The last, and also the ultimate goal, is to apply the algorithm on real databases. We are now looking at two, one is the chemical database from Eestman Chemical Company and the other is a public database on transportation.

References

[1] Aho, A.V., J.E. Hopcroft, and J.D. Ullman, "The Design and Analysis of Computer Algorithms", Addison-Wesley Publishing Company, 1974.

[2] Bisson, G., "KBG, A Knowledge Based Generalizer", The Proceedings of the 7th International Conference on Machine Learning, Morgan Kaufmann, pp. 9-15, 1990.

[3] Bisson, G., "Conceptual clustering in a first order logic representation", The Proceedings of the 10th European Conference on Artificial Intelligence, pp. 458-462, 1992.

[4] Hinton, G.E., "Learning distributed representations of concepts", Proceedings of the 8th Annual Conference of the Cognitive Science Society, 1986.

[5] Kietz, J.U. and K. Morik, "A Polynomial Approach to the Constructive Induction of Structural Knowledge", Machine Learning, 14(2):193-218, 1994.

[6] Minton, S. and I. Underwood, "Small is Beautiful: A Brute-Force Approach to Learning First-Order Formulas", Proceedings of the 12th National Conference on Artificial Intel-

ligence, pp. 168-174, 1994.

[7] Muggleton, S. and W. Buntine, "Machine invention of first order predicates by inverting resolution", Proceedings of the 5th international Machine Learning Workshop, Morgan Kaufmann, pp. 339-352, 1988.

[8] Muggleton, S., "Inverse entailment and Progol", New Generation Computing Journal, Vol. 13, pp. 245-286, 1995.

[9] Pazzani, M. and D. Kibler, "The Utility of Knowledge in Inductive Learning", Machine Learning, 9:57-94, 1992.

[10] Quinlan, J.R., "Learning Logical Definitions from Relations", Machine Learning, 5:239-266, 1990.

[11] Shen, W.-M., "Discovering regularities from Knowledge Bases", International Journal of Intelligent Systems, 7(7):623-636, 1992.

[12] Shen, W.-M., K. Ong, B. Mitbander, and C. Zaniolo, "Metaqueries for Data Mining", chapter 15. MIT Press, 1995.

[13] Stepp, R.E. and R.S. Michalski, "Inventing Goal-Oriented Classifications of Structured Objects", Machine Learning - an Artificial Intelligence Approach, vol. II, pp. 471-498, 1986.

[14] Thompson, K. and P. Langley, "Incremental concept formation with composite objects", Proceedings of the 6th International Workshop on Machine Learning, Morgan Kaufmann, pp. 373-374, 1989.

[15] Wrobe, S., "Concept Formation During Interactive Theory Revision", Machine Learning, 14(2):169-192, 1994.